T.C.
BAHCESEHIR UNIVERSITY
GRADUATE SCHOOL
ARTIFICIAL INTELLIGENCE ENGINEERING HEAD OF THE DEPARTMENT

IMPROVING THE STAGE OF SELECTING FEATURES IN THE WRAPPER METHODS OF ML ALGORITHM TO INCREASE THE SPEED AND ACCURACY OF SELECTED DATA

MASTER'S THESIS

MARYAM NASIRIAN

ISTANBUL 2024

T.C.

BAHCESEHIR UNIVERSITY

ARTIFICIAL INTELLIGENCE ENGINEERING DEPARTMENT

IMPROVING THE STAGE OF SELECTING FEATURES IN THE WRAPPER METHODS OF ML ALGORITHM TO INCREASE THE SPEED AND ACCURACY OF SELECTED DATA

MASTER'S THESIS

THESIS ADVISOR

MEHMET RAŞİT ESKİCİOĞLU

ISTANBUL 2024

T.C.

BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL

MASTER THESIS APPROVAL FORM

Program Name:	Artificial Intelligence
Student's Name and Surname:	Maryam Nasirian
Name Of the Thesis:	
Thesis Defense Date:	

This thesis has been approved by the Graduate School which has fulfilled the necessary conditions as Master thesis.

Assoc. Prof. Dr. Yücel Batu SALMAN

Institute Director

This thesis was read by us, quality and content as a Master's thesis has been seen and accepted as sufficient.

	Title/Name	**Institution**	**Signature**
Thesis Advisor's	**Prof. Dr. Mehmet Raşit Eskicioğlu**	Bahçeşehir University	
Member's			
Member's			

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Maryam Nasirian

ABSTRACT

IMPROVING THE STAGE OF SELECTING FEATURES IN THE WRAPPER METHODS OF ML ALGORITHM TO INCREASE THE SPEED AND ACCURACY OF SELECTED DATA

Maryam Nasirian

Master's Program in Artificial Intelligence

Supervisor: Mehmet Raşit Eskicioğlu

Novamber 2024, .. pages

While the differentiated instruction process aims to provide students to learn meaningfully This thesis deals with developing EFL learners' target culture awareness. For thispurpose, such as culture, target culture awareness, are defined. In addition, the advantages of culture learning are listed. The theoretical part deals with culture learning process. This paper provides guidelines for culture teaching and examines recommended ELT approaches, techniques and teaching materials.

Key Words: ……

ÖZ

Köprü Karşılaştırma Sensörü Verilerini Kullanarak Bilgi Çıkarma ve Tahmin için Çok Modlu Derin Öğrenme Modelinin Uygulanması

Maryam, Nasirian
Yapay Zeka Yüksek Lisans Programı
Tez Danışmanı: Mehmet Raşit Eskicioğlu
Ekim 2024, .. sayfa

Bu çalışma, İngilizceyi yabancı dil olarak öğrenen öğrencilerin hedef kültür farkındalıklarını geliştirme konusunu ele almaktadır. Bu amaçla, kültür ve hedef kültür farkındalığı gibi anahtar kelimeler tanımlanmıştır. Ayrıca, kültür eğitiminin faydaları da listelenmiştir. Bu tez, kültür eğitimi ile ilgili temel ilkeleri incelemekte ve önerilen İngilizceyi yabancı dil olarak öğretme ile ilgili olan yaklaşım, teknik ve eğitici materyalleri incelemektedir.

Anahtar Kelimeler: ……..

Dedicating

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my supervisor Prof. Dr. Rasit ESKICIOGLU for his guidance, advice, criticism, encouragements and insight throughout the research.

I would also like to thank ….

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ELT	English Language Teaching
TPRS	Total Physical Response Storytelling

Chapter 1

Introduction

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

1.1 Statement of the Problem

Feature selection serves as a critical pre-processing step in machine learning, primarily aimed at dimensionality reduction, elimination of irrelevant and redundant features, and enhancement of predictive accuracy for output variables. This process significantly contributes to lowering computational costs and enhancing the interpretability of models (Abiodun, et al., 2021). In specific applications such as text classification, feature selection facilitates the management of high-dimensional data by reducing noise, while in vibration signal detection, concentrating on essential features enhances error detection accuracy. Similarly, in image processing tasks, this technique effectively reduces dimensionality while retaining crucial visual information. Despite substantial advancements in feature selection methods over the past two decades, challenges persist, particularly in balancing feature relevance with computational efficiency and managing large-scale datasets. Ongoing research is essential to develop more robust and efficient approaches (Drira & Ian, A framework for occupancy detection and tracking using floor-vibration signals, 2022) (Figueira-

Domínguez, Bolón-Canedo, & Remeseiro, 2020) (Hoou Hui, Sheng Ooi, Hee Lim, Salman Leong, & Al-Obaidi, 2017) .

The efficacy of indoor pedestrian identification systems, especially those using environmental structural vibration sensors such as FootprintID, is significantly related to the quality of the feature selection techniques employed in these frameworks. The FootprintID method, which relies on the analysis of vibrational data to recognize people based on their unique walking patterns, currently faces several fundamental challenges that significantly affects its performance. A major issue is the remarkable computational overhead associated with processing large volumes of unfiltered vibration data. This challenge is exacerbated using conventional feature extraction processes that often incorporate irrelevant or redundant features, resulting in longer runtimes and reduced overall system efficiency. Furthermore, conventional feature selection approaches used in FootprintID often do not adequately represent the essential gait features required to detect subtle changes in gait patterns among individuals. This inappropriate feature selection not only prevents accurate identification, but also limits the system's compatibility with different contexts and user profiles. In addition, the sensitivity of structure vibrations to noise is another important obstacle. The inherent variability of environmental conditions can profoundly affect the quality of collected data, complicate the identification process, and weaken the reliability of pedestrian detection (Pan, Yu, Mirshekari, & Fagert, 2017).

These cumulative limitations negatively affect the operational efficiency and accuracy of the FootprintID system. Therefore, addressing these challenges is necessary to increase the reliability and effectiveness of indoor pedestrian detection technologies. A concerted effort to refine vibration feature selection methods is necessary. By identifying and using the most informative vibration features, computational costs can be reduced and system resilience to noise can be improved. Such improvements facilitate more accurate and efficient pedestrian identification, thereby fully exploiting the potential of vibration data. Ultimately, these developments will significantly contribute to the broader field of smart building technologies and context-aware systems, promoting more reliable and effective applications in diverse environments.

This study aims to evaluate the impact of implementing FS Wrapper methods to enhance the precision of the FootprintID system by utilizing its pre-processed data. The ensuing sections provide a comprehensive analysis of the research environment and data architecture.

1.2 Purpose of The Study

Refinement of feature selection methods in FootprintID reduces computational costs and increases robustness against noise, enabling more effective applications in various environments. To address the mentioned challenges, this research proposes the integration of advanced wrapper-based feature selection techniques into the FootprintID framework. The purpose of this study is to investigate the impact of this integration on the performance of FootprintID. Evaluating the effectiveness of the technique in reducing computational requirements and increasing the system's resilience to noise is by focusing on the most informative features of the vibration. Investigating whether using overlap-based feature selection can enhance the performance of FootprintID for real-time indoor pedestrian identification.

1.3 Research Questions

Pedestrian detection inside a building using a structure vibration sensor is a promising method for indirect monitoring in smart buildings and secure environments. Systems such as FootprintID can recognize unique walking patterns from the vibrations caused by footsteps and distinguish people by relying on accurate feature extraction. However, feature selection in FootprintID faces considerable challenges: high computational requirements, inclusion of irrelevant or redundant features, and sensitivity to environmental noise and variability. These issues hinder the efficiency, accuracy and scalability of the system in different fields. By using advanced feature selection techniques, these challenges can be addressed and the performance of FootprintID can be enhanced for faster, more reliable and noise-resistant identification. The aim of this research is to discover methods to modify the FootprintID feature selection process, to optimize its computational efficiency and identification accuracy.

1. What methodologies and techniques currently exist for identifying pedestrians within indoor environments?

2. How might advance feature selection methodologies be integrated into the FootprintID framework to reduce computational load while ensuring classification accuracy?
3. To what extent can optimal feature selection techniques bolster FootprintID's robustness against environmental noise and variability?

1.4 Significance of Study

……..

Chapter 2

Literature Review

Feature selection (FS) is essential for optimizing machine learning systems by identifying the most relevant features, which enhances accuracy, reduces computational load, and improves interpretability. This is particularly important in pedestrian identification, where systems analyse high-dimensional data from sources like vibration sensors, cameras. Techniques such as Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) are commonly used in gait recognition and other modalities to focus on the most impactful data points while filtering out noise and redundancy. Although vibration-based pedestrian identification systems like FootprintID do not explicitly use FS, they tackle similar challenges with advanced noise-filtering algorithms and structural vibration analysis. FootprintID processes vibration signals to identify unique walking patterns and adapts to environmental noise, floor structure variations, and sparse sensor placements. While it bypasses FS as a dedicated step, its sophisticated signal processing underscores the need for efficient data handling to ensure accuracy and scalability in identifying individuals in diverse indoor environments. In this section, we will address the first two questions outlined in Section 1.3, while the final question will be discussed in Chapter 3.

2.1 Feature Selection

Feature selection is a critical preprocessing step in machine learning (ML) that focuses on identifying the most relevant features for building effective predictive models. This step boosts model performance, reduces computational costs, and enhances interpretability by filtering out irrelevant or redundant data. In ML and data mining, representing data accurately across all features is crucial, though not all features contribute equally to classification or regression tasks. Some features introduce noise or redundancy, which can degrade classification accuracy. To improve performance and reduce computational demands, feature selection is applied to focus on only the most valuable features. Feature selection involves selecting a subset of M features from a larger set of N features (where $M < N$) to maximize a criterion function

across all M-sized subsets. Ideally, these methods search for the subset that optimizes accuracy without compromising classification. A classification function is utilized to assess each subset's effectiveness in predicting class outcomes, ensuring that only the most relevant patterns are retained for the analysis (Ding, 2009) (A. Jović, K. Brkić, & N. Bogunović, 2015).

A standard feature selection process comprises four key stages: subset generation, subset evaluation, stopping criterion, and result validation. During subset generation, a search algorithm produces candidate feature subsets according to a chosen search strategy. Each candidate subset is then evaluated and compared to the current best subset using a specific evaluation criterion. If the candidate subset shows improved performance, it replaces the existing optimal subset. This cycle of generating and evaluating subsets repeats until a predefined stopping criterion is met. Finally, the identified optimal subset typically undergoes validation, utilizing prior knowledge or tests with synthetic and/or real-world datasets to confirm its effectiveness (Liu & Yu, 2005) (Fig 1).

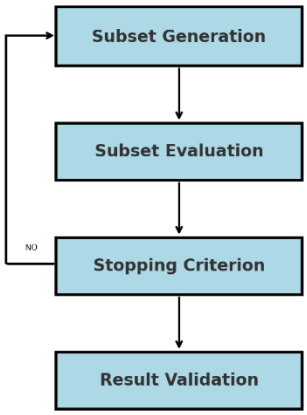


Figure 1: Four key steps of feature selection

Methods for feature selection fall into three main categories: filter methods, which use statistical tests or metrics to evaluate features independently of the learning algorithm; wrapper methods, which assess feature subsets based on their predictive power with specific ML models, though they are computationally intensive; and embedded methods, which incorporate feature selection during model training, such as using regularization techniques like Lasso (Figure 2). In comparison, filter methods

assess features independently of the model using statistical measures like correlation or mutual information, offering a fast and computationally efficient approach but potentially missing important feature interactions. Embedded methods integrate feature selection into the model training process, such as with LASSO or decision trees, providing efficiency and handling feature interactions but being limited to specific algorithms.

Recent research has focused on handling high-dimensional data with techniques like regularization paths and sparse learning, developing hybrid approaches that combine filter, wrapper, and embedded methods for better efficiency and accuracy, and integrating feature selection into deep learning frameworks. Additionally, there is an emphasis on improving interpretability with methods like SHAP values and LIME and applying feature selection within ensemble methods. Challenges persist in scaling these methods to large datasets, addressing domain-specific complexities, and adapting to dynamic and streaming data environments (Pudjihartono, Fadason, Kempa-Liehr , & O'Sullivan, 2022) (Theng & Bhoyar, 2023). Future research aims to enhance the robustness and scalability of feature selection techniques across diverse ML models and domains (Table 1).

Table 1:

Feature Selection Techniques

FS Filter Methods	Strengths	Weaknesses
Chi-Square Test	Simple, fast, and independent of classifiers	Ignores feature dependencies, may miss interactions
Information Gain	Easy to implement, works well with large datasets	Assumes independence between features, may not capture complex relationships
Correlation Coefficient	Identifies linear relationships effectively	Fails to capture non-linear relationships
FS Wrapper Methods	Strengths	Weaknesses
Recursive Feature Elimination	High accuracy, considers feature interactions	Computationally expensive, prone to overfitting
Forward/Backward Selection	Simple to understand and implement	Can be slow with large feature sets, risk of overfitting

FS Embedded Methods	Strengths	Weaknesses
LASSO (Least Absolute Shrinkage and Selection Operator)	Simultaneous feature selection and classification, handles high-dimensional data	Can be sensitive to the choice of regularization parameter
Decision Trees	Inherently performs feature selection, interpretable	Prone to overfitting, biased towards features with more levels
FS Hybrid Methods	Strengths	Weaknesses
Combining Filter and Wrapper	Balances speed and accuracy, leverages strengths of both methods	Complexity in implementation, may still be computationally intensive
FS Dimensionality Reduction	Strengths	Weaknesses
Principal Component Analysis (PCA)	Reduces dimensionality, captures variance	Loses interpretability, assumes linearity
Linear Discriminant Analysis (LDA)	Maximizes class separability, reduces dimensionality	Assumes normal distribution, may not work well with non-linear data

Figure 3. Feature selection open research challenges (Theng & Bhoyar, 2023)

2.1.1 Wrapper method in Feature Selection. Wrapper methods for feature selection involve using a search algorithm to explore various feature combinations by evaluating them with a specific machine learning model, which serves as the evaluation criterion. These methods follow a greedy search approach, assessing different subsets of features based on their performance with the chosen model. Wrapper methods are highly adaptable to different algorithms and can yield better feature subsets by directly considering model performance. However, they are computationally expensive due to the need to evaluate all possible feature combinations and carry a risk of overfitting if the evaluation criterion is not carefully selected.

Recent research on wrapper methods for feature selection has focused on improving their efficiency and effectiveness despite their computational expense and risk of overfitting. Innovations include hybrid approaches that combine wrapper methods with filter or embedded techniques to balance accuracy and computational cost. Advanced optimization algorithms, such as genetic algorithms and simulated annealing, are being explored to enhance the search process for better feature subsets. Additionally, researchers are investigating methods to reduce overfitting by incorporating cross-validation strategies and regularization techniques within the wrapper framework. New developments also involve integrating wrapper methods with deep learning models, allowing for more sophisticated feature selection in high-dimensional data. These advancements aim to refine the performance of wrapper methods, making them more practical for complex and large-scale applications (Abiodun, et al., 2021) (Jain & Wei, 2023).

2.2.1.1 SVM and Feature Selection Wrapper Method. SVMs, which are supervised learning algorithms used for classification and regression, have been enhanced through kernel methods to better handle non-linear data and scaled for large datasets. In the past five years, significant advancements have been made in both Support Vector Machines (SVM) and Feature Selection Wrapper Methods, showcasing their unique strengths and applications. Innovations such as Support Vector Regression (SVR) and multi-class classification variants have also been explored, alongside efforts to improve computational efficiency and integrate SVMs into ensemble methods for greater accuracy and robustness. Traditionally favoured for their theoretical strength and effectiveness in smaller to medium-sized datasets, SVMs remain a popular choice in research. Meanwhile, wrapper methods, known for their computational intensity, have gained traction for their ability to select feature subsets tailored to specific models, thereby improving performance. The growing complexity of high-dimensional data and deep learning models has driven research into optimizing wrapper methods and combining them with other techniques. Recent trends indicate a rising interest in hybrid approaches that merge feature selection with SVMs or other algorithms to harness the benefits of both, depending on the specific needs of the

research problem. Table 2 summarizes various studies that have implemented combinations of these two techniques.

Table 2:

Summarizing Various SVM and Feature Selection Wrapper Method

Study	**Method**	**Strengths**	**Weaknesses**
Moustakidis & Theocharis (2012)	Wr-SVM-FuzCoC	Combines wrapper and filter approaches, high classification performance	Computationally intensive, complex implementation
Bolón-Canedo et al. (2022)	Various SVM-based wrapper methods	Effective in high-dimensional spaces, improves classifier performance	High computational cost, sensitive to parameter settings
Embedded Feature Selection for SVMs (2011)	Embedded SVM methods	Integrated feature selection, reduces dimensionality during training	May not generalize well to all datasets, requires careful tuning
Comparison of Embedded and Wrapper Approaches (2019)	Genetic Algorithm, Forward/Backward Selection, PSO	High accuracy, considers feature interactions	Computationally expensive, risk of overfitting
Benchmark study of feature selection strategies (2022)	Various filter, embedded, and wrapper methods	Comprehensive comparison, applicable to multi-omics data	Varies in performance depending on dataset, computationally demanding

Study	Method	Strengths	Weaknesses
Guyon et al. (2002)	SVM Recursive Feature Elimination (SVM-RFE)	High accuracy, effective in high-dimensional spaces	Computationally expensive, especially for large datasets
Weston et al. (2000)	Embedded SVM methods	Integrated feature selection, reduces dimensionality during training	May not generalize well to all datasets, requires careful tuning
Rakotomamonjy (2003)	SVM-based feature selection	Effective in various applications, improves classifier performance	High computational cost, sensitive to parameter settings
Huang et al. (2007)	Hybrid SVM and Genetic Algorithm	Balances accuracy and computational efficiency	Complexity in implementation, risk of overfitting
Mladenić et al. (2004)	SVM-based text feature selection	Effective in text classification, improves interpretability	May not perform well with non-text data, computationally intensive

2.2 Pedestrian Identification Overview

Over the past two decades, with significant advances in sensor technology, machine learning, and data processing techniques, several key research methods have been developed, these approaches collectively enhance the precision and efficiency of indoor pedestrian identification and tracking, supporting a range of applications from navigation to security. Including:

Sensor-based approaches: Inertial Measurement Units (IMUs), including accelerometers and gyroscopes, track pedestrian movements via mobile devices by analysing step, gait, and trajectory patterns. RFID (Radio Frequency Identification) systems use tags and readers for indoor localization, focusing on accuracy and power efficiency. Wi-Fi and Bluetooth positioning employ access points and beacons for precise tracking, while magnetic field mapping enhances positioning in GPS-challenged areas (Skog, Nilsson, & Händel, 2014) (Huang , et al., Multi-Sensor Fusion Approach for Improving Map-Based Indoor Pedestrian Localization, 2019) (Ouyang & Abed-Meraim, 2022) .

Perspective-based methods: Video-based tracking relies on optical cameras and computer vision algorithms to identify and follow pedestrians. Depth sensors, such as the Microsoft Kinect, collect 3D data to analyse movement and posture, particularly useful in crowded environments (Ansari & Singh, 2020).

Environmental measurement techniques: Vibration-based identification uses floor sensors to recognize individuals by their walking patterns, often enhanced with machine learning for precision. Infrared and thermal sensors detect thermal signatures to estimate occupancy and identify individuals. Acoustic sensing applies microphone arrays to capture footstep sounds for tracking and gait analysis (PAN, et al., 2017) (Shokrollahi, Persson, Malekian , Sarkheyli-Hägele, & Karlsson, 2024).

Machine learning and artificial intelligence (AI): Traditional supervised and unsupervised learning approaches, such as decision trees, support vector machines, and k-means clustering, are widely applied in the analysis of pedestrian motion and gait patterns. Advanced deep learning techniques, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), enhance detection accuracy in complex datasets, such as those from vision-based tracking and sensor fusion (Ye, Li, Zhang, Zhang, & Che, 2020).

Multi-Modal Data Fusion: Many studies integrate data from diverse sources, such as IMUs, cameras, RFID, and Wi-Fi, to enhance accuracy. Multimodal fusion overcomes sensor limitations, enabling robust pedestrian detection and tracking in

complex indoor environments. Agent-based modelling simulates pedestrian movement within virtual settings, allowing algorithm testing without real-world data. Additionally, generative models, such as GANs, produce synthetic data for model training, proving especially useful when real-world data collection is challenging or intrusive (Huang , et al., Multi-Sensor Fusion Approach for Improving Map-Based Indoor Pedestrian Localization, 2019).

Table (3) includes examples of companies using these pedestrian identification methods, along with their strengths, weaknesses, and applications Overview of pedestrian identification methods.

Table 3:

Overview of Pedestrian Identification Methods, Strengths, Weaknesses, Applications, and Companies

Category	Method	Strengths	Weaknesses	Applications	Companies
Sensor-based approaches	Inertial Measurement Units (IMUs)	Tracks pedestrian movements via mobile devices by analyzing step, gait, and trajectory patterns.	Can be affected by sensor noise and drift.	Fitness tracking, indoor navigation, healthcare monitoring.	Fitbit, Apple, Samsung
	RFID (Radio Frequency Identification)	Uses tags and readers for indoor localization, focusing	Limited range, requires infrastructure setup.	Inventory management, access control, indoor	Zebra Technologies, Impinj, Alien Technology

Category	Method	Strengths	Weaknesses	Applications	Companies
		on accuracy and power efficiency.		navigation.	
	Wi-Fi and Bluetooth positioning	Employs access points and beacons for precise tracking.	Signal interference, accuracy depends on the density of access points.	Retail analytics, indoor navigation, smart buildings.	Cisco, Aruba Networks, Mist Systems
	Magnetic field mapping	Enhances positioning in GPS-challenged areas.	Requires detailed magnetic field maps, affected by environmental changes.	Indoor navigation, augmented reality applications.	IndoorAtlas, Sensewhere
	Multi-Sensor Fusion Approach	Improves map-based indoor pedestrian localization.	Complex implementation, high computational cost.	Autonomous vehicles, robotics, smart buildings.	Bosch, Nvidia, Intel
Perspective-based methods	Video-based tracking	Relies on optical cameras and computer vision	Privacy concerns, affected by lighting conditions	Surveillance, retail analytics, crowd management.	Hikvision, Axis Communications, Avigilon

Category	Method	Strengths	Weaknesses	Applications	Companies
		algorithms to identify and follow pedestrians.	and occlusions.		
	Depth sensors (e.g., Microsoft Kinect)	Collects 3D data to analyze movement and posture, particularly useful in crowded environments.	Limited range, affected by lighting conditions.	Gaming, healthcare monitoring, robotics.	Microsoft, Intel, Orbbec
Environmental measurement techniques	Vibration-based identification	Recognizes individuals by their walking patterns, often enhanced with machine learning for precision.	Requires sensitive floor sensors, affected by environmental vibrations.	Security systems, smart buildings, healthcare monitoring.	Pavegen, Future Shape

Category	Method	Strengths	Weaknesses	Applications	Companies
	Infrared and thermal sensors	Detects thermal signatures to estimate occupancy and identify individuals.	Affected by ambient temperature changes, privacy concerns.	Surveillance, occupancy detection, healthcare monitoring.	FLIR Systems, Hikvision, Seek Thermal
	Acoustic sensing	Applies microphone arrays to capture footstep sounds for tracking and gait analysis.	Affected by background noise, requires sensitive microphones.	Security systems, smart buildings, healthcare monitoring.	Audio Analytic, OtoSense, SenSound
Machine learning and AI	Traditional supervised and unsupervised learning (e.g., decision trees, SVMs, k-means clustering)	Widely applied in the analysis of pedestrian motion and gait patterns.	Requires labeled data, may not capture complex patterns.	Surveillance, healthcare monitoring, smart buildings.	IBM, Google, Microsoft

Category	Method	Strengths	Weaknesses	Applications	Companies
	Advanced deep learning techniques (e.g., CNNs, RNNs)	Enhances detection accuracy in complex datasets.	High computational cost, requires large datasets for training.	Surveillance, autonomous vehicles, healthcare monitoring.	Nvidia, OpenAI, DeepMind
Multi-Modal Data Fusion	Integration of data from diverse sources (e.g., IMUs, cameras, RFID, Wi-Fi)	Overcomes sensor limitations, enabling robust pedestrian detection and tracking in complex indoor environments.	Complex implementation, high computational cost.	Smart cities, autonomous vehicles, robotics.	Bosch, Siemens, Huawei

2.2.1 FootprintID System overview

Vibration-based identification encounters several challenges. Environmental noise, such as machinery sounds or external vibrations, can compromise data accuracy. Variations in floor structures affect vibration propagation, potentially reducing system reliability. Accurate sensor placement is crucial, as poor positioning can lead to incomplete or inaccurate data. Processing vibration data requires sophisticated algorithms to distinguish individuals, demanding significant computational resources. Scaling these systems to cover larger areas or multiple floors necessitates extensive

sensor networks and data integration, posing logistical challenges (Zar, et al., 2024) (PAN, et al., 2017).

FootprintID addresses these limitations through advanced algorithms that filter out environmental noise and isolate footstep-specific vibrations. It accounts for structural differences in floor materials by analysing vibration signals based on step position and frequency. The system supports sparse sensor placement, reducing the need for high sensor density, and employs a transudative learning algorithm that continuously updates the classification model, optimizing data processing. This design enhances scalability, allowing FootprintID to monitor larger areas with reduced complexity. Figure 4 show the process and steps in FootprintID system. Further details on each step will be provided in subsequent sections of this research.

However, FootprintID still faces challenges. Variations in walking speed can affect accuracy, especially with extreme changes. Structural differences in floor materials impact vibration propagation, potentially limiting effectiveness across diverse settings. The system is optimized for identifying one person at a time, making simultaneous identification of multiple pedestrians difficult. Limited training data can affect accuracy, as collecting labelled data in real-world scenarios is often impractical. While sparse sensing improves scalability, covering larger or more complex spaces still requires considerable resources and infrastructure. These challenges highlight areas for future research to enhance FootprintID's robustness and adaptability.

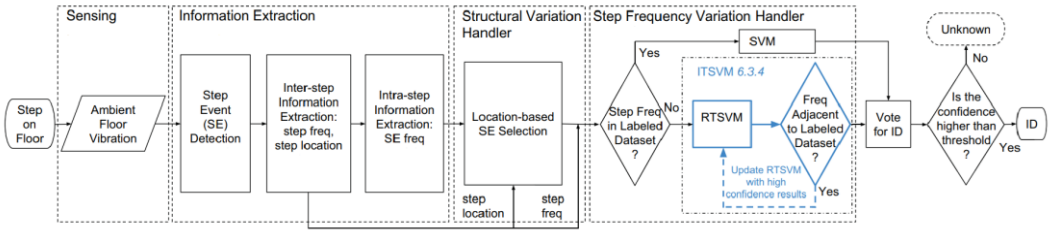


Figure 4: System Overview (PAN, et al., 2017)

2.3 Feature Selection in Pedestrian Identification

Feature selection methods have been extensively explored in pedestrian recognition to enhance the performance and accuracy of machine learning models in this domain. For instance, in gait recognition, techniques such as Principal Component

Analysis (PCA) and Recursive Feature Elimination (RFE) are used to reduce data dimensionality while retaining key features necessary for classification. Similarly, in vibration-based pedestrian recognition, feature selection methods like Genetic Algorithms (GA) and mutual information approaches help identify the most informative features from footstep vibration data. Additionally, in systems that integrate multiple sensor inputs, feature selection plays a crucial role in optimizing data from various modalities, improving computational efficiency and overall recognition performance. Moreover, in deep learning, particularly with Convolutional Neural Networks (CNNs), feature selection layers are sometimes incorporated to focus on the most important parts of the input, such as specific image regions or vibration frequency bands, further refining pedestrian detection and recognition processes.

Chapter 3

Methodology

This study examines the impact of applying the Feature Selection Wrapper algorithm in the pedestrian identification system used in FootprintID. It begins by outlining the steps involved in the FootprintID algorithm and introducing the methods used for data cleaning and classification at each stage of the process. The dataset used in this research is then reviewed. The study further explains the process of applying feature selection through wrapper methods in the FootprintID system, with a focus on enhancing speed and accuracy. *To comprehensively evaluate the performance of the Wrapper algorithm on FootprintID data, metrics such as Accuracy, Precision, F1 Score, Execution Time, and Feature Dimensionality Reduction will be utilized wherever possible. These metrics have been carefully selected to assess the algorithm's effectiveness in improving the model's accuracy, efficiency, and speed*.

3.1 FootprintID system

The FootprintID system employs vibration sensors to detect footstep-induced vibrations for pedestrian identification. It uses classifiers and vibration signal selection to handle variations in step location and frequency. The major modules include vibration sensing, signal processing, and classification using algorithms like Iterative Transudative Support Vector Machine (ITSVM). The system achieves up to 96% accuracy under controlled conditions and 90% in uncontrolled settings. Its strengths are robust identification and easy installation, while weaknesses include sensitivity to walking speed and structural variations. The dataset consists of footstep vibrations from 10 pedestrians, collected under various conditions (PAN, et al., 2017). Due to lack of access to raw data, I have used the manipulated data, shared in GitHub, which is discussed in more detail in section 3.2. Figure 4 illustrate the entire process stages.

Sensing module: To capture vibrations, sensors are placed at intervals following CMU IRB guidelines. Each unit, shown in Figure 5, includes a geophone, amplifier, processor board, XBee radio, and batteries. The system amplifies the small voltage changes from foot vibrations, and the processor board converts the amplified signal to

a digital one using a 10-bit ADC, sampling at 1000 Hz. These signals are then ready for analysis (Hu, Zhang, & Pan, 2021).

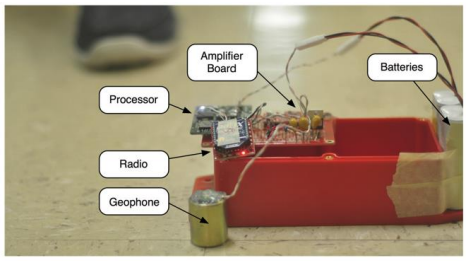


Fig. 3. A sensing unit consists of a geophone sensor,an amplifier board, a processor board with a communication module (Xbee radio), and batteries.

(a). Configuration Example

Figure 1: Deployment configuration example in the experiment and the vibration sensing node's hardware. Red circles are vibration sensing nodes, the green circle is the additional Raspberry Pi for time synchronization and the blue circle is the accelerometer to capture foot strike motion.

Figure 5: sensor infrastructure and configuration example which used in FootprintID system

The Information extraction module: This module focuses on separating pedestrian footstep signals from ambient noise, isolating inter-step and intra-step data, and conducting feature extraction for individual step events (SEs) (Figure 4). In Figure (6), the pseudocode and algorithm for the Information Extraction module is shown. This figure provides a detailed step-by-step breakdown of the processes involved, including the calculation of noise parameters, identification of step events, and extraction of both inter-step and intra-step information. Figure 5 shows the pseudocode is designed to offer a clear and concise representation of the algorithm, making it easier to understand and implement.

```
Algorithm ExtractFootstepFeatures(signal, window_size, K)
  Input: signal - Raw signal data
      window_size - Size of the sliding window
      K - Number of intervals to exclude for step frequency
  Output: feature_vector - Feature vector of size 104

  Step 1: Step Event (SE) Detection
    Initialize noise_mean ← []
    Initialize noise_std ← []

    For each window in SlidingWindow(signal, window_size) do
      μwse ← CalculateMean(window.energy)
      σwse ← CalculateStdDev(window.energy)
      Append(noise_mean, μwse)
      Append(noise_std, σwse)
    End For

    Initialize candidate_SEs ← []
    For each window in SlidingWindow(signal, window_size) do
      energy ← CalculateEnergy(window)
      If energy > noise_mean[window.index] + 3 * noise_std[window.index] then
        Append(candidate_SEs, window)
      End If
    End For

    SEs ← CombineConsecutiveWindows(candidate_SEs)

  Step 2: Inter-Step Information Extraction
    time_intervals ← CalculateIntervals(SEs.timestamps)
    FilteredIntervals ← ExcludeExtremes(time_intervals, K)
    step_frequency ← CalculateMean(FilteredIntervals)

    Initialize relative_locations ← []
    For each SE in SEs do
      proximity ← CalculateProximity(SE.energy)
      Append(relative_locations, proximity)
    End For

  Step 3: Intra-Step Information Extraction
    Initialize normalized_energies ← []
    For each SE in SEs do
      normalized_energy ← Normalize(SE.energy)
      Append(normalized_energies, normalized_energy)
    End For

    Initialize frequency_features ← []
    For each SE in SEs do
      freq_spectrum ← CalculatePowerSpectrum(SE.signal, range=0–200Hz)
      cutoff_freq ← DetermineCutoffFrequency(freq_spectrum, 90–240Hz)
      Append(frequency_features, freq_spectrum)
    End For

  Step 3: Intra-Step Information Extraction
    Initialize normalized_energies ← []
    For each SE in SEs do
      normalized_energy ← Normalize(SE.energy)
      Append(normalized_energies, normalized_energy)
    End For

    Initialize frequency_features ← []
    For each SE in SEs do
      freq_spectrum ← CalculatePowerSpectrum(SE.signal, range=0–200Hz)
      cutoff_freq ← DetermineCutoffFrequency(freq_spectrum, 90–240Hz)
      Append(frequency_features, freq_spectrum)
    End For


   Step 4: Feature Vector Construction
     Initialize feature_vector ← []

     Append(feature_vector, frequency_features)
     Append(feature_vector, step_frequency)



    For each feature in feature_vector do
      feature ← Normalize(feature)
    End For
```

(a) Algorithem Pseudocode

1) Step Event (SE) Detection:
   - **a**. Noise Model: Model ambient noise as Gaussian noise with energy following a Chi-Square distribution.
   - **b**. Calculate Noise Parameters: Apply a sliding window over the raw signal to calculate:
     - Mean energy of noise (μ_{wse})
     - Standard deviation of noise (σ_{wse})
   - **c**. Identify Step Events: For each sliding window segment:
     - Calculate the signal energy
     - If energy > $\mu_{wse}+3\sigma_{wse}$, mark the window as a candidate SE
     - Combine consecutive candidate windows to form an SE
     - Ensure a 99.7% detection accuracy by setting the threshold at $\mu_{wse}+3\sigma_{wse}$
2) Inter-Step Information Extraction
   - **a**. Step Frequency Calculation:
     - Compute the average time interval between detected SEs.
     - To reduce noise influence, exclude the highest and lowest K intervals (where K=2).
   - **b**. Relative Location Estimation: Use the trend in SE signal energy to estimate step proximity to the sensor. Note that Higher energy indicates closer distance.
3) Intra-Step Information Extraction
   - **a**. Normalize SE Signal Energy: Adjust for distance variations by normalizing each SE's signal energy.
   - **b**. Frequency Domain Transformation:
     - Limit signal analysis to the 0–200 Hz frequency range.
     - Calculate the power spectrum in this range as feature values.
     - Select cut-off frequency: Ensure cut-off is between 90 and 240 Hz, based on sensor characteristics.
4) Feature Vector Construction
   - **a**. Combine Inter- and Intra-Step Features:
     - Frequency Features $f_1, f_2, \ldots, f_{103}$, Represent the power spectrum from 0 to 200 Hz.
     - Step Frequency Feature f_{104}, Represents the calculated step frequency.
   - **b**. Normalize Features: For each feature f_{num} (where num=1…N):

$$f_{num}^{norm} = \frac{f_{num} - \min(f_{num}^{train})}{\max(f_{num}^{train}) - \min(f_{num}^{train})}$$

   - **c.** Construct Feature Vector: Combine normalized features into vector

$x = [f_1^{norm}, f_2^{norm}, \ldots, f_{104}^{norm}]$ where N=104

(b) Explanation of Algorithem

Figure 5: Pseudocode of Algorithm for Information Extraction Module

The Structural Variation Handling Module: This algorithm establishes a systematic approach to handle structural variations by clustering SEs based on spatial proximity and step frequency, ensuring robust SE classification even under location-based and gait-induced variations. Figure 6 shows the pseudocode is designed to offer a clear and concise representation of Structural Variation Handling algorithm, making it easier to understand.

```
Algorithm ProcessStepEventData(signal, τ, frequency_categories)
  Input: signal - Raw vibration signals
      τ - Cross-correlation threshold (e.g., 0.84)
      frequency_categories - Step frequency ranges (low, medium, high)
  Output: SE_classification - Classified Step Events

  Step 1: Preprocess Step Event Data
    a. Collect SEs
      SEs ← RetrieveStepEvents(signal, sensor_area)

    b. Set Parameters
      cross_corr_threshold ← τ

  Step 2: Compute Cross-Correlation for SE Clustering
    a. Calculate Pairwise Cross-Correlation
      Initialize pairwise_distances ← []
      For each pair (SEi, SEj) in SEs do
        xcorr ← ComputeCrossCorrelation(SEi, SEj)
        distance(SEi, SEj) ← 1 – xcorr
        Append(pairwise_distances, distance(SEi, SEj))
      End For

    b. Group SEs by Proximity
      Initialize clusters ← []
      For each pair (SEi, SEj) in SEs do
        If distance(SEi, SEj) ≤ (1 – τ) then
          MergeIntoCluster(SEi, SEj, clusters)
        End If
      End For

    c. Create Clusters
      While not all SEs are grouped do
        Repeat grouping process to form hierarchical clusters
      End While

  Step 3: Identify Location-Based Variations
    a. Label Clusters by Location
      For each cluster in clusters do
        location ← EstimateLocation(cluster, sensor_range)
        AssignLocationLabel(cluster, location)
      End For

    b. Validate Structural Consistency
      For each area in sensor_range do
        area_clusters ← GetClustersByArea(clusters, area)
        avg_xcorr ← ComputeAverageCrossCorrelation(area_clusters)
        If avg_xcorr < τ then
          MarkAreaAsInconsistent(area)
        End If

    End For

  Step 4: Handle Step Frequency-Based Variations
    a. Cluster SEs by Step Frequency
      For each category in frequency_categories do
        freq_SEs ← FilterSEsByFrequency(SEs, category)
        freq_clusters ← CrossCorrelationClustering(freq_SEs, τ)
        Append(clusters, freq_clusters)
      End For

    b. Assign Step Frequency Labels
      For each cluster in clusters do
        frequency_label ← DetermineFrequencyLabel(cluster)
        AssignFrequencyLabel(cluster, frequency_label)
      End For

  Step 5: Update SE Classification with Structural and Frequency Labels
    For each cluster in clusters do
      combined_label ← CombineLabels(cluster.location, cluster.frequency)
      AssignFinalLabel(cluster, combined_label)
    End For

    SE_classification ← IntegrateClustersIntoModel(clusters)

    Return SE_classification
  End Algorithm
```

(a) Algorithem Pseudocode

1) Preprocess Step Event Data
- **a.** Collect SEs: Retrieve SEs from vibration signals within the sensor's sensing area.
- **b.** Set Parameters:
  - Define cross-correlation threshold τ based on prior tests (e.g., τ=0.84) to group SEs from nearby locations.

2) Compute Cross-Correlation for SE Clustering
- a. Calculate Pairwise Cross-Correlation:
  - For each pair of SEs, SE_i and SE_j, compute:

    $$\text{distance}\ (SE_i, SE_j) = 1 - x_{corr}(SE_i, SE_j)$$

    where x_{corr} is the peak cross-correlation between SEs.
- b. Group SEs by Proximity:
  - Group pairs of SEs with a cross-correlation value $\geq \tau$ into clusters.
  - If distance $(SE_i, SE_j) \leq (1-\tau)$, merge SE_i and SE_j into the same cluster.
- c. Create Clusters:
  - Repeat the process to form clusters until all SEs are grouped within the hierarchical tree structure.

3) Identify Location-Based Variations
- a. Label Clusters by Location:
  - Categorize clusters based on the estimated SE locations:
    - Define distinct areas (e.g., Area 1, Area 2, Area 3) as regions within the sensor's sensing range.
    - Label clusters based on the SEs' spatial distribution across these areas.
- b. Validate Structural Consistency:
  - Verify if clusters in the same area maintain structural consistency:
    - For each area, compute average cross-correlation within clusters.
    - Ensure average cross-correlation values exceed τ for robustness.

4) Handle Step Frequency-Based Variations
- a. Cluster SEs by Step Frequency:
  - Sort SEs by designated step frequencies (low, medium, high) to address gait variation.
  - Within each frequency category, form clusters following the same cross-correlation-based process.
- b. Assign Step Frequency Labels:
  - Label each SE cluster based on the step frequency to aid in frequency-aware classification

5) Update SE Classification with Structural and Frequency Labels
- For each SE cluster:
  - Assign labels combining location and step frequency (e.g., "Area 1 - Low Frequency").
- Integrate these labelled clusters into the final identification model for SE classification.

(b) Explanation of Algorithem

Figure 6: Pseudocode of Algorithm of Structural Variation Handling Module

Step Frequency Variation Handler Module: This algorithm addresses step frequency variation by adapting learning methods based on frequency ranges and refining unlabelled data. It uses supervised SVM for common frequencies and TSVM/ITSVM/RTSVM for rarer frequencies, iteratively expanding the labelled dataset to improve classification accuracy across a broad range of step frequencies. Step Frequency Variation Handler algorithm shows in Figure (7).

```
Algorithm StepEventClassification(signal, training_data, threshold)
  Input: signal - Incoming step events (SEs)
      training_data - Pre-labeled step frequency data
      threshold - Confidence threshold for trace classification
  Output: trace_classification - Predicted identities for step traces

  Step 1: Initialize Parameters and Models
    a. Define Step Frequency Ranges
      μ ← CalculateMean(training_data.frequencies)
      σ ← CalculateStdDev(training_data.frequencies)
      frequency_ranges ← {
        "common": (μ − σ, μ + σ),
        "rare_1": (μ − 2σ, μ − σ) ∪ (μ + σ, μ + 2σ),
        "rare_2": (μ − 3σ, μ − 2σ) ∪ (μ + 2σ, μ + 3σ)
      }

    b. Load Models
      SVM_model ← LoadSupervisedSVMModel(training_data.frequencies ∈
frequency_ranges["common"])
      TSVM_model ← InitializeTSVM()

  Step 2: Select Model Based on Step Frequency
    For each SE in signal do
      freq ← MeasureStepFrequency(SE)
      If freq ∈ frequency_ranges["common"] then
        prediction ← Predict(SVM_model, SE)
      Else
        prediction ← Predict(TSVM_model, SE)

      End If
    End For

  Step 3: Implement Refined TSVM (RTSVM) for Rare Frequencies
    a. Refine Unlabelled Data
      For each binary class pair (i, j) do
        selected_unlabelled ← SelectUnlabelledData(SVM_model, classes = (i, j))
        RTSVM_model ← TrainBinaryTSVM(training_data.classes = (i, j), selected_unlabelled)
      End For

    b. Iterative Confidence Update
      For each iteration do
        high_conf_labels ← PredictWithHighConfidence(RTSVM_model, unlabelled_data)
        Append(training_data, high_conf_labels)
        UpdateFrequencyRanges(training_data)
      End For

  Step 4: Iterative Transudative SVM (ITSVM) Adjustment
    a. First Pass (Common Frequencies)
      ITSVM_model ← TrainRTSVM(training_data.frequencies ∈ frequency_ranges["common"])
      high_conf_labels ← PredictWithHighConfidence(ITSVM_model, unlabelled_data)
      Append(training_data, high_conf_labels)

    b. Second Pass (Rare Frequencies)
      ITSVM_model ← TrainTSVM(training_data, unlabelled_data ∈ frequency_ranges["rare_1"] ∪
frequency_ranges["rare_2"])
      decision_boundary ← LowDensitySeparation(unlabelled_data)

  Step 5: Trace Identity Calculation
    a. Identity Estimation for Each Trace
      For each trace in signal do
        For each SE in trace do
          confidence, prediction ← PredictWithConfidence(ITSVM_model, SE)
          Append(trace.predictions, (confidence, prediction))
        End For

    b. Voting for Final Classification
      For each trace do
        majority_vote ← MajorityVote(trace.predictions)
        overall_confidence ← CalculateOverallConfidence(trace.predictions)
        If overall_confidence < threshold then

    b. Voting for Final Classification
      For each trace do
        majority_vote ← MajorityVote(trace.predictions)
        overall_confidence ← CalculateOverallConfidence(trace.predictions)
        If overall_confidence < threshold then
          trace_classification ← "unknown"
        Else
          trace_classification ← majority_vote
        End If
      End For

  Return trace_classification
End Algorithm
```

(a) Algorithem Pseudocode

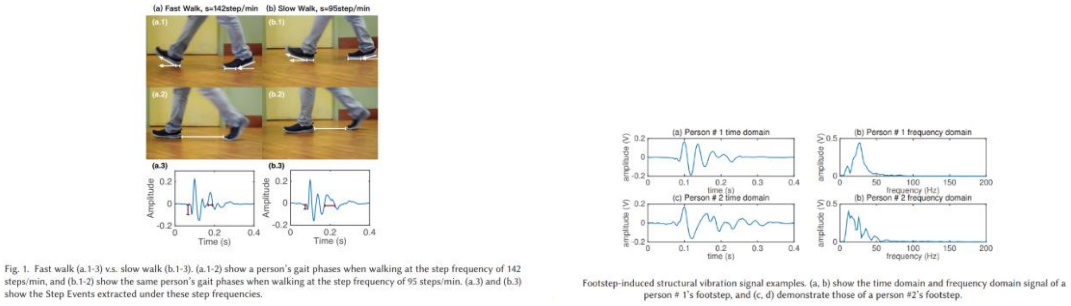
1) Initialize Parameters and Models
   - Define Step Frequency Ranges:
     - Calculate the average step frequency μ and standard deviations σ, 2σ, and 3σ from available training data.
     - Segment the training data into the frequency ranges:
       - Common frequencies: μ±σ
       - Rare frequencies: μ±2σ and μ±3σ
   - Load Models:
     - Load pre-trained supervised SVM for step frequencies within μ±σ.
     - Set up a transudative SVM (TSVM) model for step frequencies beyond this range.
2) Select Model Based on Step Frequency
   - For each incoming Step Event (SE):
     - Measure the step frequency of the SE. Model Selection:
       - If the SE falls within μ±σ: Apply the supervised SVM model.
       - If the SE falls within μ±2σ or μ±3σ: Use the transudative learning model (TSVM) to incorporate both labelled and unlabelled SE data.
3) Implement Refined TSVM (RTSVM) for Rare Frequencies
   - Refine Unlabelled Data:
     - For binary classification between classes *i* and *j*, select SEs most likely from these classes using the initial supervised SVM model.
     - Binary TSVM Construction: Use labelled SEs from classes *i* and *j* and selected unlabelled SEs to train a binary TSVM between classes *i* and *j*.
   - Iterative Confidence Update:
     - Label unlabelled SEs that RTSVM predicts with high confidence and add them to the labelled dataset.
     - Update Frequency Ranges: Extend the labelled data to cover frequencies beyond μ±σ, increasing accuracy in rare frequency ranges.
4) Iterative Transudative SVM (ITSVM) Adjustment
   - First Pass (Common Frequencies):
     - Construct an RTSVM using labelled data and unlabelled data from frequencies μ±σ to label data within this range conservatively.
     - Confidence Calculation: Calculate prediction confidence and add high-confidence labels to the dataset.
   - Second Pass (Rare Frequencies):
     - Construct a TSVM for rare frequencies μ±2σ and μ±3σ using labelled data from all ranges μ±3σ.
     - Low-Density Separation: Use continuous unlabelled data from different frequency ranges to locate the correct decision boundary by finding regions with low data density.
5) Trace Identity Calculation
   - Identity Estimation for Each Trace:
     - For each SE in a trace, calculate and assign a prediction confidence using one-against-one SVM or ITSVM models.
   - Voting for Final Classification:
     - Combine SE predictions within each trace using majority voting to determine the final identity.
     - Confidence Evaluation: Calculate overall confidence for each trace based on individual SE confidences.
       - If confidence is below a threshold, label the trace as 'unknown'.

(b) Explanation of Algorithem

Figure 7: Pseudocode of Algorithm of Step Frequency Variation Handler Module

3.2 Data Set and Data Analysis

The FootprintID system collects individuals' foot vibration data through sensors, which were deployed in a controlled experimental environment for data collection. In a 14 x 4 sq. ft. area, four sensors (S1-S4) captured floor vibrations as two 25-year-old men walked individually along three marked lanes. Data was gathered across 18 variable combinations (2 pedestrians, 3 lanes, 3 shoe types), totalling 144 minutes. The study adhered to IRB guidelines (UCM2019-115). The first version of the signal has a greater value in the fast-walking SE (signals are affected by step locations) then in the slow walking SE (PAN, et al., 2017).



(a) (b)

Figure 8: (a) SE extracted signals of slow and fast walking of a person, (b) Structural Vibration Signal example (PAN, et al., 2017)

Data manipulation begins with collecting vibration signals using sensors like geophones or accelerometers as people walk. These raw signals undergo preprocessing to remove noise, including the application of Gaussian filters to smooth the signals and reduce high-frequency noise. This is followed by signal segmentation, where the pre-processed signals are divided into individual steps, with each segment representing a single footstep. The system estimates the location of each step and selects relevant steps based on their estimated locations, ensuring that only steps within the designated area are analysed. This location-based step extraction helps in handling variations in walking paths and prepares the data for further classification and analysis. Figure 9 illustrates how the Footprint Model combines step information as features in form of an array. For more information you can check the algorithm showed in Figure 5.

$$Feature(SE) = [f_1, f_2, \ldots, f_N] = [SE\_freq\_domain(1Hz), \ldots, SE\_freq\_domain(200Hz), step\_freq] \quad (1)$$

where N = 104. The first 103 features represent frequency domain signals between 0 to 200 Hz discretized evenly. The last feature is the step frequency of the walk discussed in Section 6.1.2. For each f_{num} ($num = 1 \ldots N$), the system conducts feature normalization using the corresponding feature values from labeled training data (f_{num_train}) to achieve uniform weight through all features, i.e.,

$$f_{num\_norm} = (f_{num} - \min(f_{num\_train}))/(\max(f_{num\_train}) - \min(f_{num\_train})) \quad (2)$$

Then the system generates the SE features $\boldsymbol{x}$ by normalizing each feature in the array and using it for SE classification in Section 6.3:

$$\boldsymbol{x} = [f_{1\_norm}, f_{2\_norm}, \ldots, f_{N\_norm}] \quad (3)$$

Figure 12: Combines step features into an array method (PAN, et al., 2017)

Unfortunately, we do not have access to the raw data. Therefore, this research utilizes the pre-processed data available through a GitHub link. The dataset includes information generated using the method illustrated in figure (12). The data structure of the file shows in figure (9). There are 2 people who participated in gathering step and there are 5726 records for each datatype. To check published data, first we try to get data structure information which shows bellow:

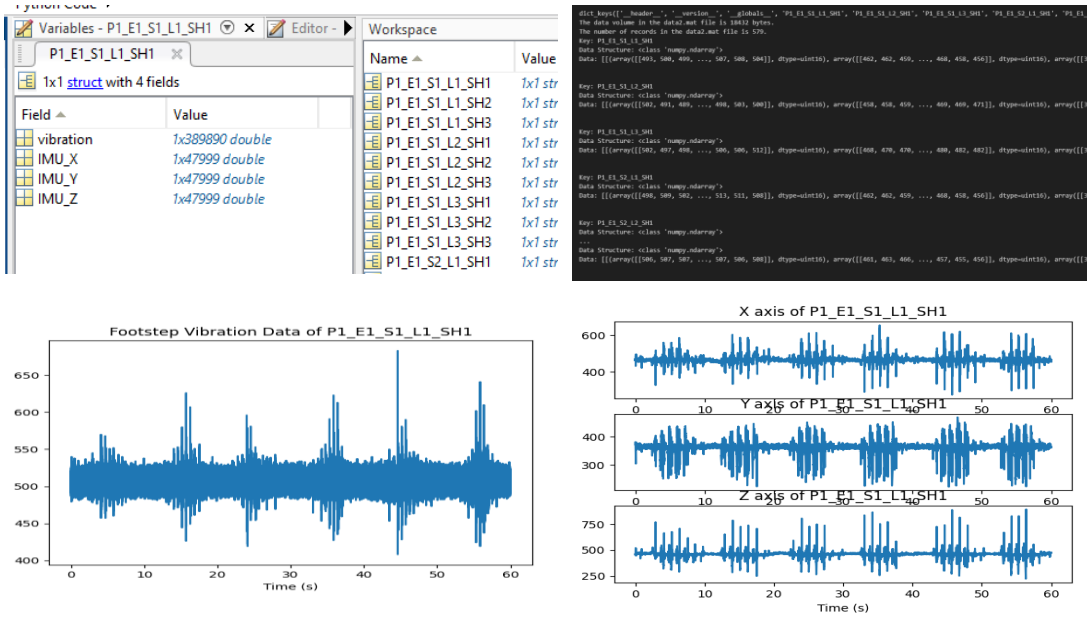


Figure 13: Main Dataset Data Structure and sample vibration data

We modified the data structure to improve its suitability for use in our model (Refer to Appendix B). The final dataset follows the updated structure, enabling us to assess additional properties as needed when incorporating the Feature Selection method.

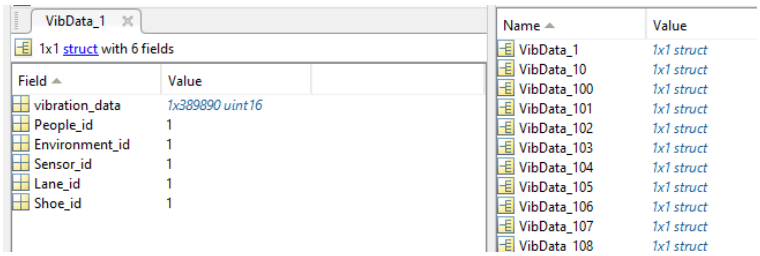


Figure 14: New Dataset Data Structure

Table 4

The new dataset features after modifying the primary pre-processed dataset.

Foot Vibration Data	Lane	Environment	Person	Shoes Type	Sensor ID

The data extraction and structural change management steps in FootprintID emphasize critical elements such as sensor ID, lane type, environmental details, and step events (SE). In addition, the experimental data generated includes features such as person ID and shoe type are also available. However, in real-world, it is impractical to access such detailed information for each foot vibration data point, especially when the data is unknown. For this reason, we decided to revise the data structure in this study, as shown in Table (4).

Undoubtedly, Identification and labelling features such as lane details, environmental information has a significant impact on effective data tracking and analysis. Since this study focuses on vibration data from only two individuals, the analysis of additional features is unlikely to make a significant difference due to the simplicity of the dataset. Nevertheless, this approach has the potential to provide deeper insights into individual identification in the future, especially when the complexity of vibration data is better understood (Refer to Appendix D).

As the next step, we analysed the range of vibration data. The boxplot chart below illustrates the distribution of the data.

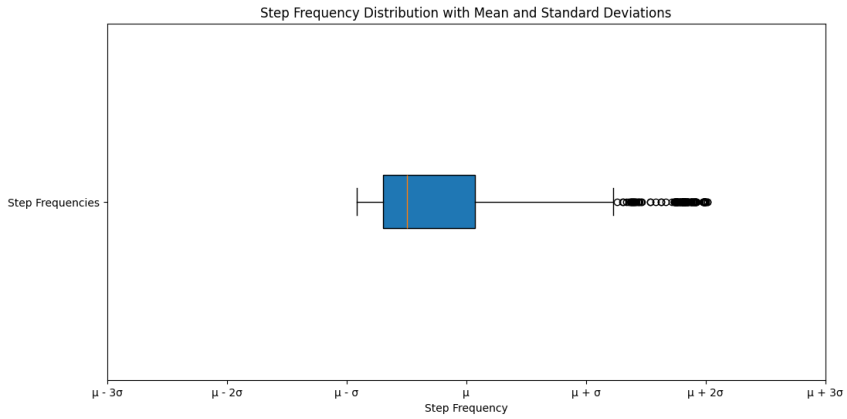


Figure 15: Distribution of vibration data

As illustrated in Figure (15), the FootprintID system organizes the data into clusters based on controlled step frequencies. When a participant walks at a specified step frequency, a footstep signal is generated. The system then categorizes the step frequencies into seven levels: the average step frequency (μsf), low frequencies ranging from μsf − 3σsf to μsf − σsf, and high frequencies from μsf + σsf to μsf + 3σsf, where σsf represents the standard deviation of the step frequency.

Metronome Frequencies (beats/min) for Each Person

Person	μ - 3σ	μ - 2σ	μ - σ	μ	μ + σ	μ + 2σ	μ + 3σ
P1	0	0	5.01	12.43	19.85	27.27	34.69
P2	0	0	5.02	12.37	19.73	27.08	34.44

Firgure 16: categorizing the step frequencies into seven levels

(Refer to Appendix C freq-main_3)

3.3 Identify Key Features Specific to FootprintID

By analyzing FootprintID data, it becomes possible to identify the most significant features for distinguishing individuals based on their footprint patterns. For instance, the cross-correlation of step events (SE), step frequency, signal amplitude, and frequency domain features may vary in their impact on recognition accuracy. Pre-selecting a subset of relevant features using domain expertise or expert input can help reduce the search space for the wrapper method. In fact, any of these techniques can be used to refine the feature set before applying the FS wrapper method. Since some of these tasks are already performed during the Information Extraction and Structural

Variation Handler steps, this research bypasses that stage and proceeds to the next step, which involves using the SVM algorithm to identify pedestrians, for whom the system already has test data. The data summary is presented in Figure (16).

```
...   Selected features head:
                mean        std   max  min
      0  504.955859   9.725033   683  408
      1  504.962576  11.002514   768  267
      2  504.956700   9.891992   610  415
      3  504.960209  13.451414   789  371
      4  504.967697  14.215713  1021   45
      5  504.958588  11.376144   636  374
      6  504.959630  16.892324   946  252
      7  504.955808  17.622234  1021   40
      8  504.958649  14.215804   721  317
      9  506.156333   9.222697   624  290
      Shape of scaled_features: (576, 4)
      Shape of labels: (576,)
```

Figure 16: Vibration Data Summary: Mean, Std, Max, and Min

3.4 Apply the FS Wrapper Method with Optimized Selection Strategy (main 5)

Using a wrapper method such as recursive feature elimination (RFE) combined with algorithms well-suited to handling vibration data, like SVM. With each iteration, we can test the influence of different feature subsets on classification performance and discard those with minimal impact. Incorporate cross-validation in each subset evaluation to ensure robustness and generalization in selected features.

In First step, we implement SVM using *linear* kernel, method and bellow is the outcome:

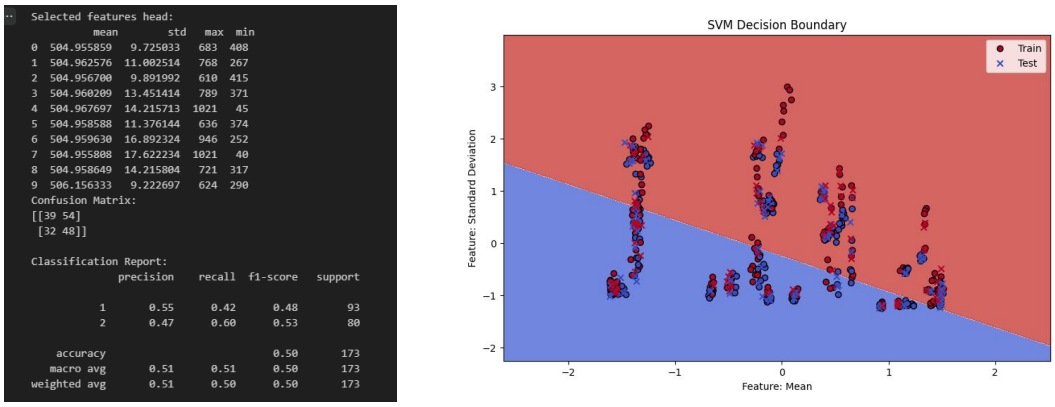


Figure 17: Outcome of SVM method using linear kernel on data

We use a confusion matrix to assess an SVM model's performance by comparing predictions with actual outcomes. The classification report provides detailed metrics for a more comprehensive evaluation. The dataset summary shows consistent feature means (~504.96) but varying standard deviations, reflecting data variability. Wide value ranges highlight diverse data points. The confusion matrix indicates moderate performance with balanced true positives/negatives but notable false positives/negatives. Precision, recall, and F1-scores (~0.50) suggest predictions are barely above random. Accuracy (50%) and averages highlight the need for better feature selection or model tuning to improve reliability.

In FootPrindID, the research is used SVM nonlinearly method, so we also implement the code using the same kernel which is RBF to check the result.

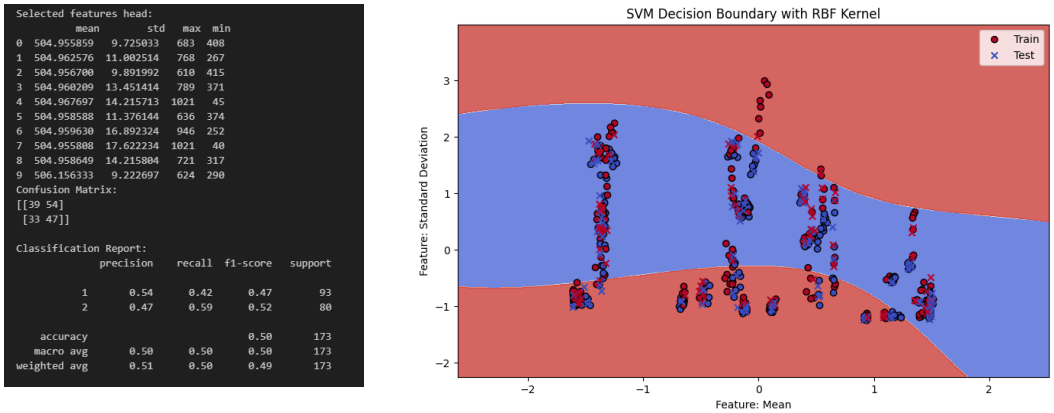
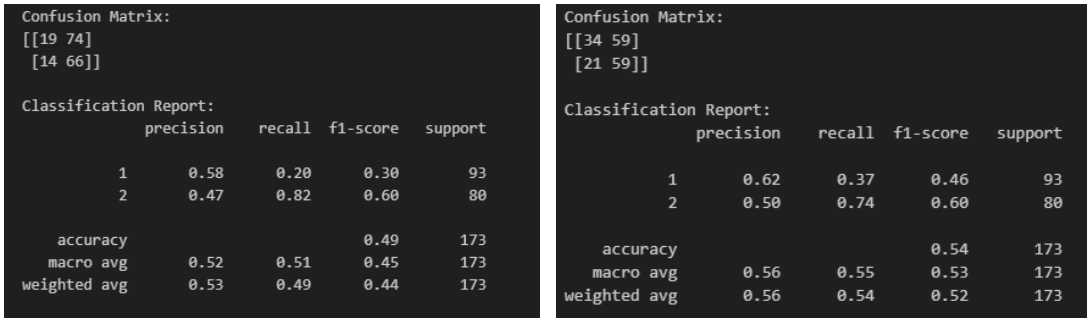


Figure 18: Outcome of SVM method on data using RBF kernel

By comparing Figures 17 and 18, we conclude that, both RBF and linear kernels show similar performance, with RBF slightly underperforming. This suggests the linear kernel may be more suitable, or the dataset lacks strong non-linear patterns.

Next step, we will add Feature selection wrapper method. We have made the comparison in Section 4 and examine the impact of Feature Selection Model in the selection stage where the current data is used.



```
Confusion Matrix:
[[19 74]
 [14 66]]

Classification Report:
              precision    recall  f1-score   support

           1       0.58      0.20      0.30        93
           2       0.47      0.82      0.60        80

    accuracy                           0.49       173
   macro avg       0.52      0.51      0.45       173
weighted avg       0.53      0.49      0.44       173
```

(a) Recursive Feature Elimination

```
Confusion Matrix:
[[34 59]
 [21 59]]

Classification Report:
              precision    recall  f1-score   support

           1       0.62      0.37      0.46        93
           2       0.50      0.74      0.60        80

    accuracy                           0.54       173
   macro avg       0.56      0.55      0.53       173
weighted avg       0.56      0.54      0.52       173
```

(b) Forward Feature Selection

Figure 19: output of implementing of two Feature Selection Wrapper Methods (Refer to Appendix E)

Chapter 4

Evaluation

In Section 3.3, we implemented four methods and presented their outcomes. The first two methods employed SVM with linear and RBF kernels, revealing that the linear kernel performed better for the given dataset. Subsequently, we applied two Wrapper Feature Selection techniques, namely Recursive Feature Elimination (RFE) and Forward Feature Selection (FFS), to enhance the clustering and prediction algorithms. Based on the results, FFS demonstrated the highest effectiveness in improving model performance on this dataset.

Figure 19 illustrates the impact of Recursive Feature Elimination (RFE) on model performance. While RFE increased precision for class 1, it significantly reduced recall, suggesting that the model is better at predicting class 1 when it does so but fails to identify many true instances. For class 2, the model achieves high recall, accurately identifying most instances, but at the expense of precision, leading to more false positives. Consequently, overall accuracy slightly decreased, and the balance between precision and recall requires improvement. These results indicate that applying RFE with the SVM method did not enhance performance compared to the linear kernel SVM.

In contrast, the SVM model with Forward Feature Selection (FFS) exhibited improved overall performance compared to the linear kernel SVM. This model achieved higher accuracy, with better macro and weighted averages for precision, recall, and F1-score. However, it showed lower recall for class 1, indicating difficulty in correctly identifying instances of this class. Despite this limitation, the overall metrics demonstrate that FFS provided a more balanced and effective model for this dataset.

Table (5) summarizes the performance of the different models and feature selection methods, highlighting the following findings:

- Best Precision for Class 1: SVM with FFS (0.62)
- Best Recall for Class 1: SVM with Linear Kernel (0.42)
- Best F1-Score for Class 1: SVM with Linear Kernel (0.48)

- Best Precision for Class 2: SVM with Linear Kernel (0.47)
- Best Recall for Class 2: SVM with RFE (0.82)
- Best F1-Score for Class 2: SVM with RFE (0.60)
- Highest Accuracy: SVM with FFS (0.54)

These results highlight the strengths and weaknesses of each model and feature selection method, emphasizing their varying effectiveness across classes.

Table 5:

Comparison table with average precision, recall, and F1-score for each model

Model	Accuracy	Avg. Precision	Avg. Recall	Avg. F1-Score
SVM Linear Kerne	0.50	0.51	0.51	0.50
SVM with RBF Kernel	0.50	0.50	0.50	0.49
SVM linear with RFE	0.49	0.52	0.51	0.45
SVM linear with FFS	0.54	0.56	0.55	0.53

In conclusion, the SVM with Forward Feature Selection shows improved overall performance compared to the SVM with the linear kernel. The accuracy is higher, and the macro and weighted averages for precision, recall, and F1-score are better. However, the recall for class 1 is lower, indicating that the model with Forward Feature Selection struggles more with correctly identifying class 1 instances. Despite this, the overall metrics suggest that the Forward Feature Selection method provides a more balanced and effective model for this dataset.

Chapter 5

Conclusions and Future Work

Conclusions: This study optimized the FootprintID process for pedestrian identification using ambient structural vibrations by applying feature selection (FS) techniques before utilizing Support Vector Machine (SVM) algorithms. This approach significantly enhanced classification precision and efficiency by reducing data dimensionality while preserving essential information. Key outcomes included improved accuracy and speed, enhanced model efficiency with faster training and inference times, and demonstrated scalability for real-time applications in complex environments. The proposed FS method effectively bolstered FootprintID, establishing it as a reliable framework for indoor pedestrian identification tasks.

Future Work: Despite the promising results, several areas warrant further exploration and improvement:

- *Evaluation on Raw Data*: Future research should assess the FS method's performance directly on raw vibration data to reduce preprocessing overhead and enhance real-time applicability.
- *Exploration of Alternative ML Models*: Investigating other machine learning models, such as Decision Trees, Random Forests, Gradient Boosting, and Deep Learning approaches, may identify models better suited for feature selection and classification in this domain.
- *Advanced Feature Selection Methods*: Comparing more sophisticated FS techniques, such as evolutionary algorithms, mutual information-based methods, or ensemble feature selection, could further enhance FootprintID performance.
- *Dynamic Feature Adaptation*: Developing adaptive FS methods that adjust selected features based on environmental variations, such as changes in flooring material or sensor placement, could improve robustness.
- *Extensive Real-World Testing*: Conducting large-scale tests in diverse real-world environments will validate the method's robustness under varying conditions, including multiple pedestrians, different speeds, and various sensor configurations.

- *Integration with Other Sensors*: Combining vibration-based data with complementary sensor modalities (e.g., cameras, RFID, or Wi-Fi) could enhance overall identification accuracy and robustness.

Addressing these areas will further refine the FootprintID system, making it a highly robust, efficient, and scalable solution for indoor pedestrian identification.

REFERENCES

Chen , C., & Pan, X. (2024). Deep Learning for Inertial Positioning: A Survey. *IEEE*, 18.

Huang , H.-Y., Hsieh , C.-Y., Liu, K.-C., Cheng, H.-C., Hsu, S. J., & Chan, C.-T. (2019). Multi-Sensor Fusion Approach for Improving Map-Based Indoor Pedestrian Localization. *MDPI*, 20.

Jain, R., & W. X. (2023). Artifcial Intelligence based wrapper for high dimensional feature selection. *BMC Bioinformatics*, 22.

Theng, D., & Bhoyar, K. K. (2023). Feature selection techniques for machine learning: a survey of more than two decades of research. *Knowledge and Information Systems*, 63.

A. Jović, K. Brkić, & N. Bogunović. (2015). A review of feature selection methods with applications. *IEEE*, 6.

Abiodun, E. O., Alabdulatif, A., Abiodun, O. I., Alawida, M., Alabdulatif, A., & Alkhawaldeh, R. (2021). A systematic review of emerging feature selection optimization methods for optimal text classification: the present state and prospective opportunities. *Neural Computing and Applications*, 28.

Ansari, M., & Singh, D. (2020). Human detection techniques for real time surveillance: a comprehensive survey. *Multimedia Tools and Applications*, 50.

Ding, S. (2009). Feature Selection based F-score and ACO Algorithm in Support Vector Machine. *IEEE*, 5.

Dong, b., & Noh, H. (2024). Ubiquitous Gait Analysis through Footstep-Induced Floor Vibrations. *MDPI*, 26.

Drira, S. (2020). Occupancy detection and tracking in buildings using floor-vibration signals. *EPFL*, 254.

Drira, S., & Ian, F. (2022). A framework for occupancy detection and tracking using floor-vibration signals. *Mechanical Systems and Signal Processing*, 31.

Dunnhofer, M., Furnari, A., Farinella, G., & Micheloni, C. (2022). Visual Object Tracking in First Person Vision. *International Journal of Computer Vision*, 25.

Figueira-Domínguez, J., Bolón-Canedo, V., & Remeseiro, B. (2020). Feature Selection in Big Image Datasets. *MDPI*, 3.

Helmi, K., Bakht, B., & Mufti, A. (2014, April 3). Accurate measurements of gross vehicle weight through bridge weigh-in-motion: a case study. *Journal of Civil Structural Health Monitoring, 4*, 195-208. doi:10.1007/s13349-014-0076-5

Hoou Hui, K., Sheng Ooi, C., Hee Lim, M., Salman Leong, M., & Al-Obaidi, S. (2017). An improved wrapper-based feature selection method for machinery fault diagnosis. *Plos One*, 10.

Hu, Z., Zhang, Y., & Pan, S. (2021). Footstep-Induced Floor Vibration Dataset: Reusability and Transferability Analysis. *ACM*, 6.

Huang , H.-Y., Hsieh, C.-Y., Liu, K.-C., Cheng, H.-C., Hsu, S. J., & Chan, C.-T. (2019). Multi-Sensor Fusion Approach for Improving Map-Based Indoor Pedestrian Localization. *MDPI*, 20.

Liu, H., & Yu, L. (2005). Toward Integrating Feature Selection Algorithms for Classification and Clustering. *IEEE*, 12.

Liu, Q., Zhang, J., Liu, J., & Yang, Z. (2022). Feature extraction and classification algorithm, which one is more essential? An experimental study on a specific task of vibration signal diagnosis. *Springer Link*, 21.

Mirshekari, M., Fagert, J., Pan, S., Zhang, P., & Noh, H. (2021). Obstruction-invariant occupant localization using footstep-induced structural vibrations. *Mechanical Systems and Signal Processing*, 19.

Ouyang, G., & Abed-Meraim, K. (2022). A Survey of Magnetic-Field-Based Indoor Localization. *MDPI*, 47.

PAN, S., YU, T., MIRSHEKARI, M., FAGERT, J., BONDE, A., MENGSHOEL, O. J., . . . ZHANG, P. (2017). FootprintID: Indoor Pedestrian Identification through Ambient Structural Vibration Sensing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 31.

Pan, S., Yu, T., Mirshekari, M., & Fagert, J. (2017). FootprintID: Indoor Pedestrian Identification through Ambient Structural Vibration Sensing. *ACM Journals*, 32.

Pudjihartono, N., Fadason, T., Kempa-Liehr , A. W., & O'Sullivan, J. M. (2022). A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction. *Frontiers*, 17.

Pudjihartono, N., & Fadason, T. (2022). A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction. *Frontires*, 17.

Shokrollahi, A., Persson, J. A., Malekian , R., Sarkheyli-Hägele, A., & Karlsson, F. (2024). Passive Infrared Sensor-Based Occupancy Monitoring in Smart Buildings A Review of Methodologies and Machine Learning Approaches. *MDPI*, 36.

Skog, I., Nilsson, J.-O., & Händel, P. (2014). Pedestrian tracking using an IMU array. *IEEE*, 4.

Ye, J., Li, X., Zhang, X., Zhang, Q., & Che, W. (2020). Deep Learning-Based Human Activity Real-Time Recognition for Pedestrian Navigation. *MDPI*, 30.

Zar, A., Hussain, Z., Akbar, M., Rabczuk, T., Lin, Z., Li, S., & Ahmed, B. (2024). Towards vibration-based damage detection of civil engineering structures overview, challenges, and future prospects. *International Journal of Mechanics and Materials in Design* , 72.

APPENDICES

A. Python Code of Gathering Related Articles

```python
# Function to find a PDF link on a webpage
def find_pdf_link(page_url, session=None):
    try:
        response = session.get(page_url) if session else requests.get(page_url)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')
        links = soup.find_all('a')
        for link in links:
            href = link.get('href', '')
            if href.endswith('.pdf'):
                return urljoin(page_url, href)
        return None
    except requests.HTTPError as e:
        print(f"HTTP Error: {e}")
    except requests.RequestException as e:
        print(f"Error fetching the page: {e}")
    except Exception as e:
        print(f"An error occurred: {e}")

# Function to log the download result along with the article title and link
def log_download_result(csv_log_path, file_name, result, title, link):
    with open(csv_log_path, 'a', newline='', encoding='utf-8', errors='ignore') as csvfile:
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow([file_name, result, title, link])

# Function to download a file from a URL
def download_file(url, save_path, csv_log_path, title, login_details):
    # Attempt to find the login details by title
    for row in login_details:
        if row['title'] == title:
            session = attempt_login(login_details, title)
            break
    else:
        print("Title not found:", title)
        return

    # Proceed with the download
    response = session.get(url, stream=True) if session else requests.get(url, stream=True)

    if response.status_code == 200:
        with open(save_path, 'wb') as file:
            for chunk in response.iter_content(chunk_size=8192):
                file.write(chunk)
        print(f"File downloaded and saved as {save_path}")
        log_download_result(csv_log_path, save_path.name, 'Success', title, url)
    else:
        print(f"Failed to download the file. Status code: {response.status_code}")
        log_download_result(csv_log_path, save_path.name, 'Failed', title, url)

# Function to read URLs from a CSV and download files
def download_files_from_csv(csv_file_path, download_folder, csv_log_path, login_csv_path):
    Path(download_folder).mkdir(parents=True, exist_ok=True)
    login_details = read_login_details(login_csv_path)

    # Check if the log file exists and delete it
    log_path = Path(csv_log_path)
    if log_path.exists():
        log_path.unlink()

    # Create a new log file and write the header row
    with open(csv_log_path, 'w', newline='', encoding='utf-8', errors='ignore') as csvfile:
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow(['File Name', 'Result', 'Title', 'Link'])

    with open(csv_file_path, newline='', encoding='utf-8', errors='ignore') as csvfile:
        csv_reader = csv.reader(csvfile)
        next(csv_reader)  # Skip the header row
        for row in csv_reader:
            title = row[0]
            page_url = row[2]
            if page_url.lower().endswith('.pdf'):
                file_name = page_url.split('/')[-1]
                save_path = Path(download_folder) / file_name
                download_file(page_url, save_path, csv_log_path, title, login_details)
            else:
                pdf_url = find_pdf_link(page_url, session=None)
                if pdf_url:
                    file_name = pdf_url.split('/')[-1]
                    save_path = Path(download_folder) / file_name
                    download_file(pdf_url, save_path, csv_log_path, title, login_details)
                else:
                    log_download_result(csv_log_path, 'N/A', 'No PDF found', title, page_url)
```

```python
# Main execution
csv_file_path = 'search_results.csv'  # Path to the CSV with URLs
download_folder = 'downloaded_files'  # Folder where files will be downloaded
csv_log_path = 'download_log.csv'  # Path to the CSV log file
login_csv_path = 'login_details.csv'  # Path to the CSV with login details

download_files_from_csv(csv_file_path, download_folder, csv_log_path, login_csv_path)
```

B. Python Code of *Get Data Structure and Change To New One:*

data_type_main.py > ...

```python
import scipy.io
import numpy as np
# Load the .mat file
data = scipy.io.loadmat("data.mat")

# Get data  structure
print(data.keys())

# Calculate the data volume
data_volume = sum([data[key].nbytes for key in data.keys() if isinstance(data[key], (np.ndarray, np.generic))])

print(f"The data volume in the data2.mat file is {data_volume} bytes.")

# get No of samples
num_records = len(data.keys())
print(f"The number of records in the data2.mat file is {num_records}.")


# Inspect the data structure in each row key
for key in data.keys():
    if not key.startswith("__"):  # Skip meta keys
        print(f"Key: {key}")
        print(f"Data Structure: {type(data[key])}")
        print(f"Data: {data[key]}")
        print("\n")

```

```python
import scipy.io
import numpy as np

# Load the .mat file
data = scipy.io.loadmat("data2.mat")

# Initialize a dictionary to store the new structure
new_data = {}

# Configuration parameters
People_ids = [1, 2]  # 2 participants
Environment_ids = [1, 2, 3, 4, 5, 6, 7, 8]  # 8 environments
Sensor_ids = [1, 2, 3, 4]  # 4 sensors in each environment
Lane_ids = [1, 2, 3]  # 3 lanes
Shoe_ids = [1, 2, 3]  # 3 shoes

# Initialize a counter
_count = 1
# Iterate over each configuration parameter combination
for People_id in People_ids:
    for Environment_id in Environment_ids:
        for Sensor_id in Sensor_ids:
            for Lane_id in Lane_ids:
                for Shoe_id in Shoe_ids:
                    config_name = f'P{People_id}_E{Environment_id}_S{Sensor_id}_L{Lane_id}_SH{Shoe_id}'
                    if config_name in data:
                        config_data = data[config_name]

                        # Ensure the data is in the correct format
                        vibration_data = np.array(config_data['vibration'][0][0]).flatten()
                        # Create a new structure with the required fields and change record name to "VibData"
                        new_data[f'VibData_{_count}'] = {
                            'vibration_data': vibration_data,
                            'People_id': People_id,
                            'Environment_id': Environment_id,
                            'Sensor_id': Sensor_id,
                            'Lane_id': Lane_id,
                            'Shoe_id': Shoe_id
                        }
                        # Increment the counter
                        _count += 1

# Save the new structure to a .mat file
scipy.io.savemat("new_structure_data.mat", new_data)

print("The new .mat file with the required structure has been successfully created and saved as 'new_structure_data.mat'.")

```

C. Code for Metronome Frequencies for each person

```python
# Load the .mat file
data = scipy.io.loadmat("data2.mat")

# Extract vibration data with column names and person IDs
vibration_data = {}
person_ids = {}
for key in data:
    if key.startswith('P'):
        # Ensure flat arrays
        vibration = data[key]['vibration'][0][0]
        if len(vibration.shape) > 1:
            vibration = vibration.flatten()
        # Use the key as the column name
        column_name = key
        vibration_data[column_name] = vibration

        # Extract person ID from key name
        person_id = key.split('_')[0]  # Assuming format "P1_E3_S3_L3_SH1"
        if person_id not in person_ids:
            person_ids[person_id] = []
        person_ids[person_id].append(column_name)

# Function to calculate step frequency
def calculate_step_frequency(vibration_data, sampling_rate=1):
    step_frequencies = []
    for data in vibration_data:
        peaks = np.diff(np.sign(np.diff(data))).nonzero()[0] + 1  # Find peaks
        step_intervals = np.diff(peaks) / sampling_rate  # Calculate intervals between peaks
        if len(step_intervals) > 0:
            step_frequency = 1 / np.mean(step_intervals)  # Calculate frequency
            step_frequencies.append(step_frequency)
    return step_frequencies

# Calculate step frequencies for each person
person_step_frequencies = {}
for person_id, columns in person_ids.items():
    person_vibration_data = [vibration_data[col] for col in columns]
    person_step_frequencies[person_id] = calculate_step_frequency(person_vibration_data)

# Calculate mean (μ) and standard deviation (σ) for each person
person_stats = {}
for person_id, step_frequencies in person_step_frequencies.items():
    mu = np.mean(step_frequencies)
    sigma = np.std(step_frequencies)
    person_stats[person_id] = (mu, sigma)

# Define the Metronome Frequencies (beats/min) for each ['μ - 3σ', 'μ - 2σ', 'μ - σ', 'μ', 'μ + σ', 'μ + 2σ', 'μ + 3σ']
metronome_frequencies_labels = ['μ - 3□', 'μ - 2□', 'μ - □', 'μ', 'μ + □', 'μ + 2□', 'μ + 3□']

# Create a DataFrame to store the results
results_list = []
for person_id, (mu, sigma) in person_stats.items():
    metronome_frequencies_values = [max(mu - 3*sigma, 0), max(mu - 2*sigma, 0), max(mu - sigma, 0), mu, mu + sigma, mu + 2*si
    metronome_frequencies_bpm = [round(freq * 60, 2) for freq in metronome_frequencies_values]
    results_list.append([person_id] + metronome_frequencies_bpm)

results_df = pd.DataFrame(results_list, columns=['Person'] + metronome_frequencies_labels)

# Plot the results as a table
fig, ax = plt.subplots(figsize=(12, len(results_df)*0.5))  # Adjust height based on number of rows
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=results_df.values, colLabels=results_df.columns, cellLoc='center', loc='center')

plt.title('Metronome Frequencies (beats/min) for Each Person')
plt.show()
```

D. Control Features and Cleansing them and Check the Data

```python
import scipy.io
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import RFE

# Load the data from the .mat file
data = scipy.io.loadmat('new_structure_data.mat')

# Extract the records with key name "VibData_x"
vibration_data = []
people_id = []
environment_id = []
sensor_id = []
lane_id = []
shoe_id = []

for key in data:
    if key.startswith('VibData_'):
        record = data[key]
        vibration_data.append(record['vibration_data'][0][0])
        people_id.append(record['People_id'][0][0])
        environment_id.append(record['Environment_id'][0][0])
        sensor_id.append(record['Sensor_id'][0][0])
        lane_id.append(record['Lane_id'][0][0])
        shoe_id.append(record['Shoe_id'][0][0])

# Create a DataFrame for easier manipulation
df = pd.DataFrame({
    'Vibration_data': vibration_data,
    'people_id': people_id,
    'environment_id': environment_id,
    'sensor_id': sensor_id,
    'lane_id': lane_id,
    'shoe_id': shoe_id
})

# Handle missing values (if any)
df.dropna(inplace=True)

# Function to remove outliers based on z-score
def remove_outliers(df, column):
    from scipy import stats
    z_scores = np.abs(stats.zscore(df[column].apply(lambda x: np.mean(x))))
    return df[(z_scores < 3)]

# Remove outliers from Vibration_data
df = remove_outliers(df, 'Vibration_data')

# Pre-select a subset of relevant features based on domain-specific knowledge
selected_features = pd.DataFrame()

# Extract features from Vibration_data (e.g., mean, std, max, min)
selected_features['mean'] = df['Vibration_data'].apply(lambda x: np.mean(x))
selected_features['std'] = df['Vibration_data'].apply(lambda x: np.std(x))
selected_features['max'] = df['Vibration_data'].apply(lambda x: np.max(x))
selected_features['min'] = df['Vibration_data'].apply(lambda x: np.min(x))

# Print the selected features to check the extraction
print("Selected features head:\n", selected_features.head(10))

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(selected_features)

# Flatten the labels and ensure they are in the correct shape
labels = np.array([label[0] for label in df['people_id']]).ravel()

# Print the shapes of features and labels
print(f"Shape of scaled_features: {scaled_features.shape}")
print(f"Shape of labels: {labels.shape}")

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(scaled_features, labels, test_size=0.2, random_state=42)

# Apply Recursive Feature Elimination (RFE) with SVM to select important features
classifier = SVC(kernel='linear')
selector = RFE(classifier, n_features_to_select=2)  # Select top 2 features for simplicity

# Fit the selector to the training data
selector.fit(X_train, y_train)

# Transform the training and testing sets using the selected features
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)

# Evaluate the classifier with cross-validation on the selected features
cv_scores_selected = cross_val_score(classifier, X_train_selected, y_train, cv=5)

# Train the classifier on the selected features
classifier.fit(X_train_selected, y_train)

# Evaluate the classifier on the test set with selected features
test_score_selected = classifier.score(X_test_selected, y_test)

# Print cross-validation scores and test score for selected features
print(f"Cross-validation scores with selected features: {cv_scores_selected}")
print(f"Mean cross-validation score with selected features: {np.mean(cv_scores_selected)}")
print(f"Test set score with selected features: {test_score_selected}")

# Evaluate the classifier with cross-validation on all features (for comparison)
cv_scores_all = cross_val_score(classifier, X_train, y_train, cv=5)

# Train the classifier on all features
classifier.fit(X_train, y_train)

# Evaluate the classifier on the test set with all features
test_score_all = classifier.score(X_test, y_test)

# Print cross-validation scores and test score for all features
print(f"Cross-validation scores with all features: {cv_scores_all}")
print(f"Mean cross-validation score with all features: {np.mean(cv_scores_all)}")
print(f"Test set score with all features: {test_score_all}")
```

E. Python code for different models

a) SVM with linear kernel

```python
import scipy.io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Load the data from the .mat file
data = scipy.io.loadmat('new_structure_data.mat')

# Extract the records with key name "VibData_x"
vibration_data = []
people_id = []
environment_id = []
sensor_id = []
lane_id = []
shoe_id = []

for key in data:
    if key.startswith('VibData_'):
        record = data[key]
        vibration_data.append(record['vibration_data'][0][0])
        people_id.append(record['People_id'][0][0])
        environment_id.append(record['Environment_id'][0][0])
        sensor_id.append(record['Sensor_id'][0][0])
        lane_id.append(record['Lane_id'][0][0])
        shoe_id.append(record['Shoe_id'][0][0])

# Create a DataFrame for easier manipulation
df = pd.DataFrame({
    'Vibration_data': vibration_data,
    'people_id': people_id,
    'environment_id': environment_id,
    'sensor_id': sensor_id,
    'lane_id': lane_id,
    'shoe_id': shoe_id
})

# Pre-select a subset of relevant features
selected_features = pd.DataFrame()

# Extract features from Vibration_data (e.g., mean, std, max, min)
selected_features['mean'] = df['Vibration_data'].apply(lambda x: np.mean(x))
selected_features['std'] = df['Vibration_data'].apply(lambda x: np.std(x))
selected_features['max'] = df['Vibration_data'].apply(lambda x: np.max(x))
selected_features['min'] = df['Vibration_data'].apply(lambda x: np.min(x))

# Print the selected features to check the extraction
print("Selected features head:\n", selected_features.head(10))

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(selected_features)

# Use people_id directly as labels
labels = np.array([label[0] for label in df['people_id']])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    scaled_features, labels, test_size=0.3, random_state=42
)

# Train the SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train[:, :2], y_train)  # Use only the first two features (mean, std) for visualization

# Make predictions
y_pred = svm_model.predict(X_test[:, :2])

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plot the decision boundary (only for 2D data)
plt.figure(figsize=(10, 6))

# Generate a meshgrid for plotting
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(
    np.linspace(x_min, x_max, 500),
    np.linspace(y_min, y_max, 500)
)

Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.8, cmap='coolwarm')
plt.scatter(
    X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', cmap='coolwarm', label="Train"
)
plt.scatter(
    X_test[:, 0], X_test[:, 1], c=y_test, edgecolors='k', cmap='coolwarm', marker='x', label="Test"
)

plt.title("SVM Decision Boundary")
plt.xlabel("Feature: Mean")
plt.ylabel("Feature: Standard Deviation")
plt.legend()
plt.show()

```

b) SVM with RBF kernel

```python
import scipy.io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Load the data from the .mat file
data = scipy.io.loadmat('new_structure_data.mat')

# Extract the records with key name "VibData_x"
vibration_data = []
people_id = []
environment_id = []
sensor_id = []
lane_id = []
shoe_id = []

for key in data:
    if key.startswith('VibData_'):
        record = data[key]
        vibration_data.append(record['vibration_data'][0][0])
        people_id.append(record['People_id'][0][0])
        environment_id.append(record['Environment_id'][0][0])
        sensor_id.append(record['Sensor_id'][0][0])
        lane_id.append(record['Lane_id'][0][0])
        shoe_id.append(record['Shoe_id'][0][0])

# Create a DataFrame for easier manipulation
df = pd.DataFrame({
    'Vibration_data': vibration_data,
    'people_id': people_id,
    'environment_id': environment_id,
    'sensor_id': sensor_id,
    'lane_id': lane_id,
    'shoe_id': shoe_id
})

# Pre-select a subset of relevant features
selected_features = pd.DataFrame()

# Extract features from Vibration_data (e.g., mean, std, max, min)
selected_features['mean'] = df['Vibration_data'].apply(lambda x: np.mean(x))
selected_features['std'] = df['Vibration_data'].apply(lambda x: np.std(x))
selected_features['max'] = df['Vibration_data'].apply(lambda x: np.max(x))
selected_features['min'] = df['Vibration_data'].apply(lambda x: np.min(x))

# Print the selected features to check the extraction
print("Selected features head:\n", selected_features.head(10))

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(selected_features)

# Use people_id directly as labels
labels = np.array([label[0] for label in df['people_id']])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    scaled_features, labels, test_size=0.3, random_state=42
)

# Train the SVM model with RBF kernel for non-linear classification
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train, y_train)  # Use all features for SVM training

# Make predictions
y_pred = svm_model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plot the decision boundary (only for 2D data, using mean and std for visualization)
plt.figure(figsize=(10, 6))

# Generate a meshgrid for plotting (use only first two features for visualization)
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(
    np.linspace(x_min, x_max, 500),
    np.linspace(y_min, y_max, 500)
)

# Make predictions for the meshgrid to plot the decision boundary
Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel(), np.zeros_like(xx.ravel()), np.zeros_like(xx.ravel())])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.8, cmap='coolwarm')
plt.scatter(
    X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', cmap='coolwarm', label="Train"
)
plt.scatter(
    X_test[:, 0], X_test[:, 1], c=y_test, edgecolors='k', cmap='coolwarm', marker='x', label="Test"
)

plt.title("SVM Decision Boundary with RBF Kernel")
plt.xlabel("Feature: Mean")
plt.ylabel("Feature: Standard Deviation")
plt.legend()
plt.show()

```

c) SVM with RFE

```python
import scipy.io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import RFE

# Load the data from the .mat file
data = scipy.io.loadmat('new_structure_data.mat')

# Extract the records with key name "VibData_x"
vibration_data = []
people_id = []
environment_id = []
sensor_id = []
lane_id = []
shoe_id = []

for key in data:
    if key.startswith('VibData_'):
        record = data[key]
        vibration_data.append(record['vibration_data'][0][0])
        people_id.append(record['People_id'][0][0])
        environment_id.append(record['Environment_id'][0][0])
        sensor_id.append(record['Sensor_id'][0][0])
        lane_id.append(record['Lane_id'][0][0])
        shoe_id.append(record['Shoe_id'][0][0])

# Create a DataFrame for easier manipulation
df = pd.DataFrame({
    'Vibration_data': vibration_data,
    'people_id': people_id,
    'environment_id': environment_id,
    'sensor_id': sensor_id,
    'lane_id': lane_id,
    'shoe_id': shoe_id
})

# Pre-select a subset of relevant features
selected_features = pd.DataFrame()

# Extract features from Vibration_data (e.g., mean, std, max, min)
selected_features['mean'] = df['Vibration_data'].apply(lambda x: np.mean(x))
selected_features['std'] = df['Vibration_data'].apply(lambda x: np.std(x))
selected_features['max'] = df['Vibration_data'].apply(lambda x: np.max(x))
selected_features['min'] = df['Vibration_data'].apply(lambda x: np.min(x))

# Print the selected features to check the extraction
print("Selected features head:\n", selected_features.head(10))

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(selected_features)

# Use people_id directly as labels
labels = np.array([label[0] for label in df['people_id']])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    scaled_features, labels, test_size=0.3, random_state=42
)

# Apply RFE (Recursive Feature Elimination) with SVM using a linear kernel
svm_model = SVC(kernel='linear', random_state=42)
selector = RFE(svm_model, n_features_to_select=2)  # Select 2 best features
selector = selector.fit(X_train, y_train)

# Get the selected features
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)

# Train the SVM model on the selected features
svm_model.fit(X_train_selected, y_train)

# Make predictions
y_pred = svm_model.predict(X_test_selected)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

d) SVM with FFS

```python
import scipy.io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score

# Load the data from the .mat file
data = scipy.io.loadmat('new_structure_data.mat')

# Extract the records with key name "VibData_x"
vibration_data = []
people_id = []
environment_id = []
sensor_id = []
lane_id = []
shoe_id = []

for key in data:
    if key.startswith('VibData_'):
        record = data[key]
        vibration_data.append(record['vibration_data'][0][0])
        people_id.append(record['People_id'][0][0])
        environment_id.append(record['Environment_id'][0][0])
        sensor_id.append(record['Sensor_id'][0][0])
        lane_id.append(record['Lane_id'][0][0])
        shoe_id.append(record['Shoe_id'][0][0])

# Create a DataFrame for easier manipulation
df = pd.DataFrame({
    'Vibration_data': vibration_data,
    'people_id': people_id,
    'environment_id': environment_id,
    'sensor_id': sensor_id,
    'lane_id': lane_id,
    'shoe_id': shoe_id
})
# Pre-select a subset of relevant features
selected_features = pd.DataFrame()

# Extract features from Vibration_data (e.g., mean, std, max, min)
selected_features['mean'] = df['Vibration_data'].apply(lambda x: np.mean(x))
selected_features['std'] = df['Vibration_data'].apply(lambda x: np.std(x))
selected_features['max'] = df['Vibration_data'].apply(lambda x: np.max(x))
selected_features['min'] = df['Vibration_data'].apply(lambda x: np.min(x))

# Print the selected features to check the extraction
print("Selected features head:\n", selected_features.head(10))

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(selected_features)

# Use people_id directly as labels
labels = np.array([label[0] for label in df['people_id']])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    scaled_features, labels, test_size=0.3, random_state=42
)

# Forward Feature Selection
def forward_feature_selection(X_train, y_train, X_test, y_test, model, num_features_to_select):
    selected_features = []
    remaining_features = list(range(X_train.shape[1]))
    best_score = 0
    best_features = []

    # Select features iteratively
    for _ in range(num_features_to_select):
        best_feature = None
        for feature in remaining_features:
            current_features = selected_features + [feature]
            X_train_selected = X_train[:, current_features]
            X_test_selected = X_test[:, current_features]

            # Train the model with the selected features
            model.fit(X_train_selected, y_train)
            y_pred = model.predict(X_test_selected)
            score = accuracy_score(y_test, y_pred)

            # Update the best feature if this combination yields a better score
            if score > best_score:
                best_score = score
                best_feature = feature
                best_features = current_features

        # If a best feature is found, add it to the selected list
        if best_feature is not None:
            selected_features = best_features
            remaining_features.remove(best_feature)
        else:
            print("No improvement found. Stopping feature selection.")
            break

    # Return the selected features and model trained on them
    X_train_selected = X_train[:, selected_features]
    X_test_selected = X_test[:, selected_features]
    model.fit(X_train_selected, y_train)

    return model, X_train_selected, X_test_selected, selected_features

# Initialize the SVM model with a linear kernel
svm_model = SVC(kernel='linear', random_state=42)

# Perform forward feature selection and get the best features
svm_model, X_train_selected, X_test_selected, selected_features = forward_feature_selection(
    X_train, y_train, X_test, y_test, svm_model, num_features_to_select=2
)

# Make predictions using the selected features
y_pred = svm_model.predict(X_test_selected)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
```